SN Computer Science (2021) 2:35
https://doi.org/10.1007/s42979-020-00420-8

ORIGINAL RESEARCH

An Ontology-Based Approach to Automated Test Case Generation

Shreya Banerjee[1] · Narayan C. Debnath[1] · Anirban Sarkar[2]

Received: 21 September 2020 / Accepted: 3 December 2020 / Published online: 12 January 2021

Abstract

Software testing is as old as software itself. However, the techniques, tools, and processes used by researchers to ensure product quality are constantly evolving. Application of knowledge management technologies in automated test case generation is one of them. This paper addressed the issue of ontology-based automated test case generation in the case of black box testing. In this context, several challenges are present in existing literature. The prime challenges among are (1) major approaches are confined to a specific domain, (2) least consideration about modified domain knowledge, (3) lack of methodology for auto-identification of pre-conditions and different combinations among test input data and (4) poor requirements and domain coverage. The proposed methodology, in this paper, is aimed to resolve these issues by devising a rule-based reasoner that can auto generate the test cases. The proposed method takes an ontology-based requirements specification as an input. The novelty of the proposed method is the specification of domain independent inference rules based on which the devised reasoner can generate test cases for different domains and systems automatically. This contribution of the proposed work facilitates in improving both user's requirements coverage and domain coverage. The devised reasoned, in this paper, is implemented in Apache Jena (Apache Jena, https://jena.apache.org., Accessed 2020/09/04). In addition, the usability of the proposed work is illustrated using a suitable case study.

Keywords Automated test case · Test case ontology · Rule-based reasoner · Test case generation tool

Introduction

Software Testing is a time-consuming and resource-hungry task that depends on advanced expert knowledge. Researchers are continuously seeking to develop new approaches to address this issue [1]. In modern days, the process of software testing is performed using systematic test activities, such as test planning and design, visual reviews of requirements documents and program code, program testing, system testing, acceptance testing, and so on [3]. Despite all these efforts, errors are remain undetected in the code. According to CapgeminiWorld Quality Report 2018–19, the budget allocation for quality assurance and testing, as percentage of IT expenditures in the software industry, has come down in recent years but still accounted for 26% in 2018 [3]. This issue requires serious attention on automated testing tools and techniques.

Despite successful achievements in automation on script execution and white-box testing, there is still a lack of automation of black-box testing of functional requirements [3]. Tedious manual process of test case generation for black-box testing largely depends upon domain knowledge [12]. Usually, in black-box testing, test cases are formed by looking at different users' requirements. However, it requires 40–70% of the software test life cycle that has affected on cost, time and effort factors due to the frequent changes in requirements and having different terminologies [4]. In this context, requirement-based testing can be used to uncover faults and defects in artefacts during early stage development

This article is part of the topical collection "Applications of Software Engineering and Tool Support" guest edited by Nabendu Chaki, Agostino Cortesi and Anirban Sarkar.

Shreya Banerjee
shreya.banerjee@eiu.edu.vn

Narayan C. Debnath
narayan.debnath@eiu.edu.vn

Anirban Sarkar
sarkar.anirban@gmail.com

[1] Department of Software Engineering, Eastern International University, Thu Dao Mot City, Binh Duong Province, Vietnam

[2] Department of Computer Science and Engineering, National Institute of Technology, Durgapur, India

[5]. Further, to aid automated generation of test cases from requirements, requirements specification should be represented precisely.

These days, knowledge management is extensively used in software testing and influences software testing processes, methods and models [2]. Test-case generation using available system knowledge is one of the crucial applications of knowledge-based software testing among many. Usually, the testing process requires collaboration between several stakeholders [11]. This creates the necessity that domain experts need to be able to communicate with the testers effectively. Ontology is defined as explicit specification of shared conceptualization [7]. It can represent concepts and relationships within a domain in a way that allows automated reasoning [7]. Ontologies are considered as an enabling technology for representation and sharing of domain knowledge in software testing. It can be used to represent users' requirements precisely. Automated reasoning on ontology specification can be accomplished using of inference rules [6]. An ontology can represent requirements from a software requirements specification, and the inference rules can describe strategies for deriving test cases from that ontology [4].

However, several challenges are present in related existing approaches those have applied ontology for automating their test cases. The crucial challenges among those are, first, most of approaches are confined to specific applications. Those cannot be applied over different domain and applications. Test specifications need to be represented in high level of abstractions, so that those can be further reused over different implementations [5]. Second, automated generation of test data is required to deal with domain knowledge that is changing continuously. Third, in several test cases, pre-conditions have high impact on test input. Pre-conditions represents the context, in which the test need to be executed. Thus, based on pre-conditions, result of a similar test input can vary. Hence, automated extraction of pre-conditions from requirements specification is also a significant task. Fourth, sub test cases can be generated from different combinations of test data reside in single test case. It can help in update or addition of new test cases. Hence, automated extraction of different relationships among test data is required.

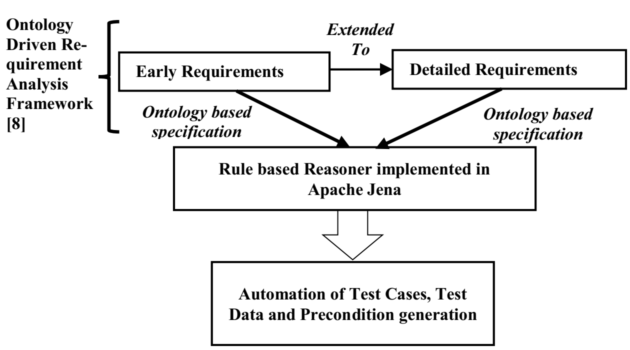

Fig. 1 Overview of the proposed approach for ontology-based automation of test case generation

This paper is aimed to address these aforementioned challenges using an automated testing approach. In the proposed approach, a rule-based reasoner is devised that will automatically create test cases based on an ontological representation of requirements described in [8]. The contribution of the proposed method are many. First, the ontology-based test specification and proposed rules are domain independent. Hence, the rules can be applied to different applications on same or different domain for automating the test cases. Thus, the proposed approach aids in customised domain- or application-specific test case generation. In this way, the proposed approach has addressed the first challenge. Second, since the proposed approach is based on an ontolog-based description of requirements specification, it can infer new knowledge from existing domain knowledge and synthesize test data. This contribution can deal with continue modifications of domain knowledge. Thus, the second challenge is addressed in the proposed methodology. Third, both pre-conditions of test data and different association among test data are identified in the proposed method. Thus, the approach is capable to identify test context and the different conditions of test execution automatically. Fourth, proposed method in this paper is an effort to devise a software testing tool that can save time and cost by accomplishing automated test case generation for customized domains or applications in the context of black-box testing. Thus, the proposed approach aids in obtaining domain or system coverage.

With these objectives, the paper is organized in the following way. Section 2 has described the related work. Section 3 has proposed the methodology for devising the rule-based reasoned. Section 4 has implemented the proposed methodology in Apache Jena [10]. Further, the proposed approach is illustrated using a case study in Sect. 5. Finally, Sect. 6 has concluded the paper.

Related Work

Few approaches exist in the literature those have applied ontology in automated test case generation for black-box testing. In [3], authors have described an approach that has automated complete testing process using ontologies and inference rules. The approach takes an ontology-based software requirements specifications as input and produces test scripts as output. However, in the described requirements ontology, associations among different requirements are not considered. Further, the prescribed inference rules are domain specific. Hence, inference rules are need to be

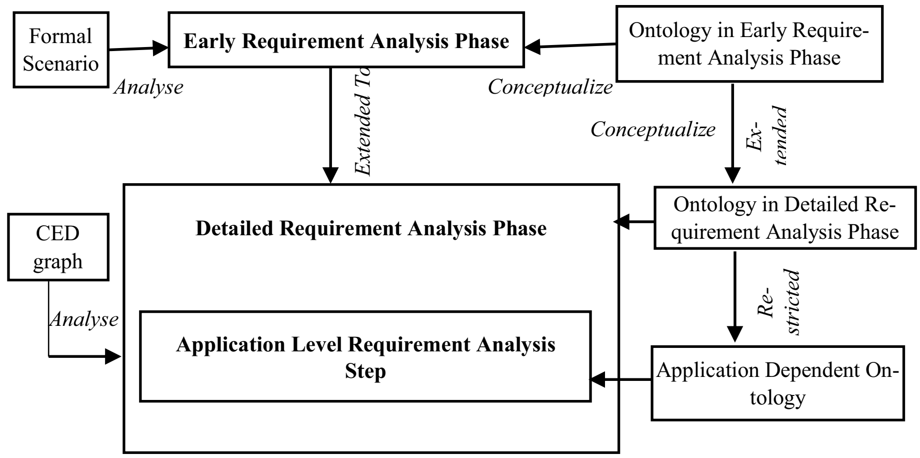

Fig. 2 Detail illustration of Ontology Driven Requirement Analysis Framework [8]

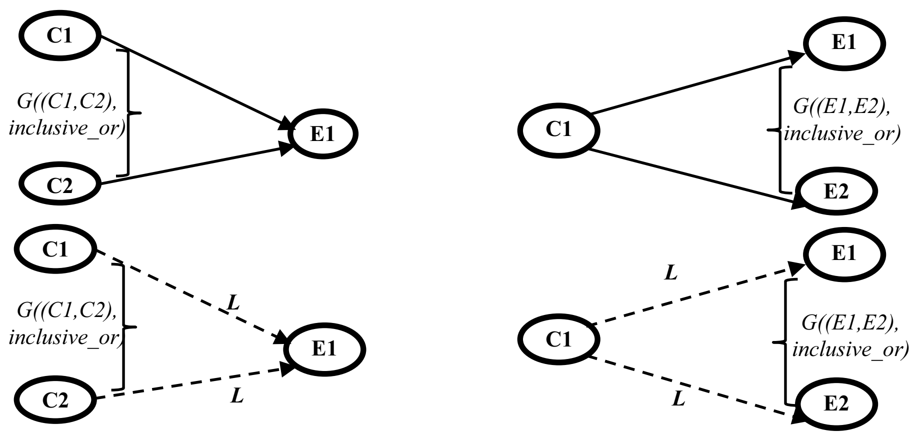

Fig. 3 Different Variations of Inclusive Or conditions based on Transformation and Dependency Relationships; **a** All causes have transformed towards the effect; **b** One cause has transformed towards all of the effects; **c** All of the causes has dependent on one effect; **d** One cause has dependent on all of the effects. [8]

formulated manually for different domains and applications. In [14], a general knowledge-based test case generation framework is described that allows customized definition of domain and system specific coverage criteria. However, this approach has not considered pre-conditions, test scenario and relationships among different requirements. In [12], an automatic test case generation framework is devised that involves ontology-based requirement specification and learning-based methods for conducting black box testing. This method also integrates ontology-based system with learning-based testing algorithm to automate generation of test cases, test execution and test verdict construction. However, the described method is presented only from conceptual perspective. Authors have not developed the framework in practice. Further, they have not considered about different combinations of test data in a test case. In [15, 16], the described ontology is intended for automating of test cases for web-services. Thus, these approaches are specific to certain domain. Likewise, in [17], authors have described the method for automated test case generation of multi-agent systems. In [18], authors has developed a Reference Ontology on Software Testing (ROoST). This ontology establishes a common conceptualization about the software testing domain, such as defining a common vocabulary for knowledge workers with respect to the testing domain, structuring testing knowledge repositories, annotating testing knowledge items, and for making search for relevant information easier. Authors have described about ontology testing but they have not prescribed about ontology-based test case generation.

Majority of the existing approaches focus on specific domain for developing their automated test case generation framework. Thus, the domain or system coverage criteria for their approaches are very limited. However, in this paper, a general framework is proposed, that can facilitate automated generation of test cases for different domain and application. Thus, the domain coverage criteria of the proposed approach is good. Besides that, few approaches have specified about automated preconditions and different relationships among test input. Both these artefacts are required to grasp the test context. Further, these artefacts also aid in deriving new test cases from exiting one and update test cases. The proposed approach, in this paper, has facilitated in automated generation of both pre-conditions and test input.

Table 1 Summarization of proposed test case ontology and equivalent ODRA facets

Proposed test case ontology	Corresponding ODRA facets
TestOutput	Effects extended from a specific user goal in detailed requirements analysis phase
TestInput	Causes transformed to the effects extended from a specific user goal
PreCondition	Effects on which causes are dependent
	Effects with which one effect is related using Require guard functions

Proposed Methodology

Proposed approach in this paper is accomplished based on the outline illustrated in Fig. 1. The main objective of the proposed work is to devise a reasoner that can automate test case generations. The devised reasoner takes ontology-based requirements specification as input. A set of inference rules are proposed to build the proposed reasoner. The reasoner has generated test cases along with pre-conditions and expected result based on those inference rules. Section 3.1 has summarized the description of ODRA. Section 3.2 has specified the proposed method of the reasoner. Further, Sect. 3.3 has proposed the different inference rules.

Brief Description of ODRA (Ontology Driven Requirements Analysis Framework) [8]

ODRA described in [8] is a generalized requirements engineering framework that can be applied towards different domains and applications. ODRA is specified for both early and detailed requirements analysis phase.

In early requirements analysis phase, the framework has represented and analyzed users' requirements based on users' goals, roles, and corresponding scenarios. Users' goals can be achieved by sequence of functionalities (F) those are resulting in real-world effects (E). Functionalities can be realized through distinct combinations between tasks, activities, user inputs, events, and other entities. Real-world effects can be specified as a set of effects. Thus, sequences of functionalities F and corresponding effects E can be represented as a scenario. Identified Goals are satisfied through E. Thus, a scenario aids to achieve Goals effectively.

In detailed requirements analysis phase, Cause-Effect-Dependency graph (CED Graph) [13] is used for analysis of users' requirements in detailed way. CED graph has represented and analyzed users' requirements from six views—Who, What, Why, When, Where and How (5W1H). In this phase, ontology has two concepts—causes and effects. Causes are equivalent to functionalities identified in early requirement analysis Phase. Cause can be defined as a set of input entities bringing changes in a domain. Effects are equivalent to a set of effects created through scenarios of functionalities in early requirement analysis Phase. Thus, effects aid in satisfying of users' Goals. Effect can be defined as a set of output states, those are created from a combination of Causes. Causes are related to effects using two crucial relationships—Transformation Relationship (*TR*) and Dependency Relationship (*DR*). Further, causes are connected with each other using different guard functions. Likewise, effects are also connected with each other using different guard functions. Those different guard functions are *And*, *Or*, *Mask*, *Inclusive_Or*, *Exclusive_Or*, *Not*, and *Require*. The detailed requirements analysis phase comprises two steps. The first step represents domain-specific requirements. The second step specifies application level requirements. Thus, the first step represents domain level causes and effects and the second step specifies application level causes and effects. ODRA was implemented Protégé [9]. The Fig. 2 has illustrated the ODRA framework. Figure 3 has demonstrated *Inclusive_Or* guard function in causes.

Proposed Methodology for Rule-Based Reasoner

A method is proposed in this section for devising the rule-based reasoner. The devised reasoner takes ODRA specification of a certain domain as input. The reasoner also takes the list of inference rules proposed in Sect. 3.3 as an input. Since, ODRA can be customized for different domains and applications; the reasoner is able to generate test cases based on different domains and applications. Besides that, the reasoner starts its execution with specific user goal id. Hence, it is able to automate the scenario related to the specific user goal. Thus, pre-conditions and different relationships among user requirements along with test data are auto generated by the proposed reasoner. Distinct guard function present in ODRA facilitates in realization of different combinations among users requirements. It also assists in test case generation as per customized requirements rather than generation of all test cases for whole requirements specification.

In the proposed method, a test case ontology is automated from the scenario related to the input goal id. The proposed set of inference rules are applied on a scenario related to a specific user goal, populate the test case

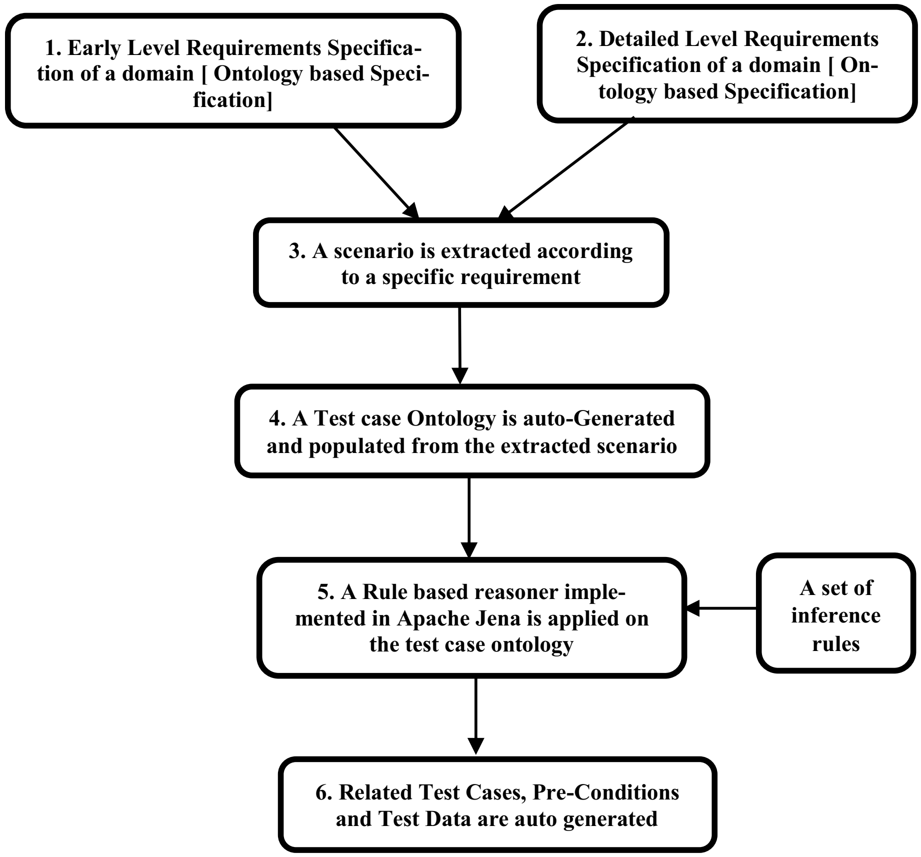

Fig. 4 A workflow model of the proposed methodology

```
IF a is an instance of concept DomainLevelCause
   b is an instance of concept DomainLevelCause
   c is an instance of concept DomainLevelEffect
   d is an instance of concept DomainLevelEffect
   l is an instance of concept DomainLevelEffect
   i is an instance of concept DomainLevelEffect
   Transformation_Realationship is a relationship
   Dependancy_Relationship is a relationship
   AND is a relationship
   Require is a relationship
   a Transformation_Relationship c
   b Transformation_Relationship c
   a AND b
   notEqual(a,b)
   a Dependancy_Relationship d
   b Dependancy_Relationship l
   c Require i
---------------------------------------------------
Then, make a as an instance of concept TestInput
      make b as an instance of concept TestInput
      make c as an instance of concept TestOutput
      make d as an instance of concept PreCondition
      make l as an instance of concept PreCondition
      make i as an instance of concept PreCondition
      create the statement c hasTestInput a
      create the statement c hasTestInput b
      create the statement c hasPrecondition d
      create the statement c hasPrecondition l
      create the statement c hasPrecondition i
```

Fig. 5 Example of an domain independent Inference Rule that facilitates in automated of Test Cases

Table 2 List of domain level causes and effects in the example specified in Sect. 3.3

Domain level causes	Domain level effects (TR/DR relationships)
Order is received (C1)	Order is confirmed (E1)(TR)
Check stock availability (C2)	
Create invoice (C3)	Order is confirmed (E1) (DR)
Update the stock (C4)	
Create invoice (C3)	Ship the good (E3) (TR)
Update the stock (C4)	
Packing the good (C5)	Order is confirmed (E1) (DR)
	the good is packaged (E2) (TR)
	Ship the good (E3) (TR)

ontology and generate the required test cases. This test case ontology includes 3 classes and 2 object relationships. The three classes are "PreCondition", "TestInput" and "TestOutput". Further, the object relationships are "hasPrecondition" and "hasTestInput". Different facets of the automated test case ontology is identified from different concepts and relationships of ODRA. Table 1 has summarized this mapping. Further, the auto-generated test cases based on this automated test case ontology include three segments. Those

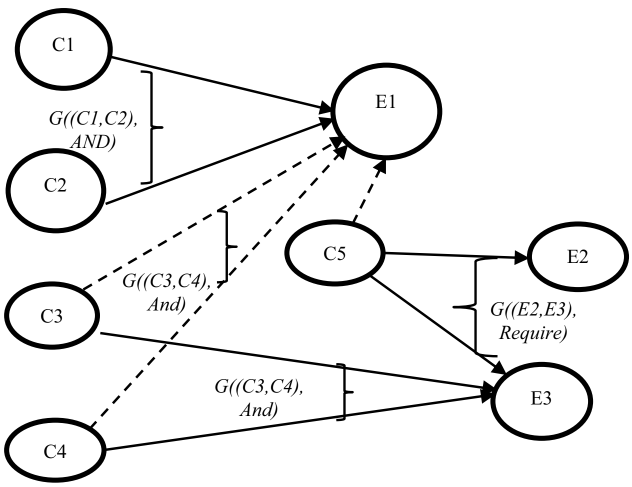

Fig. 6 CED graph of the example specified in Sect. 3.3

are "Pre Condition", "Test Input" and "Expected Result". Method 1 has specified the proposed step-wise method of devising the rule-based reasoner. Further, in Sect. 4, this stepwise algorithm is implemented in Apache Jena. Figure 4 has specified a workflow diagram of the proposed methodology.

Table 3 Mapping from domain level causes and effects related to the example to facets in the proposed test case ontology using an inference rule

Domain level causes / Effects	Instances in the rule specified in Fig. 5	Facets in the proposed test case ontology
C3	*a*	TestInput
C4	*b*	TestInput
E1	*d*	PreCondition
E3	*c*	TestOutput
E1	*l*	PreCondition
E2	*i*	PreCondition

Method 1:	Proposed method for devising the rule-based reasoner for automating test cases
Input:	Ontology-based requirements specification of a certain domain, a specific user requirement, set of inference rules
Output:	Test cases along with pre-conditions and test data for the input user requirement
Step 1:	Read the file *F* related to the input ontology specification
Step 2:	Create a ontology file *T* for the test case ontology
Step 3:	Read the goal id of a specific user requirement
Step 4:	Print the objective of the user requirement to the console
Step 5:	Create a list effect_List
Step 6:	Store all the domain level effects extended from that requirement in the list effect_List
Step 7:	Create a class "DomainLevelEffect" and write it to the file T. Add all the elements of the list effect_List to this class as instances.
Step 8:	Find all the domain level causes from file *F* those are transferred towards the domain level effects through transformation relationship
Step 9:	Create a class "DomainLevelCause" and write it to the file *T*. Add all the domain level causes found in the previous step as instances to the class "DomainLevelCause".Create an object property Transformation relationship and write it to the file *T*. Write the statements containing identified domain level causes, transformation relationship, corresponding effects to the file *T*.
Step 10:	Find all the relationships among these domain level causes from file *F* through six guard functions such as And, Inclusive_Or, Exclusive_Or, Mask, Not, Require.
Step 11:	Create six object properties "AND", "Require", "Inclusive_Or", "Exclusive_Or", "Mask", "Not". Write all these object properties to file *T*. Write all the statements containing identified source domain level cause, guard functions, target domain level cause to the file *T*.
Step 12:	Find all the effects those are dependent from these domain level causes from file *F*.
Step 13:	Add all the effects identified in Step 12 as instances to the class "DomainLevelEffect" in file *T*.
Step 14:	Create an object property dependency relationship and write it to the file *T*. Write all the statements containing identified domain level causes, dependency relationship, identified domain level effects.
Step 15:	Find all the guard functions present among domain level effects identified in step 12. Write the statements containing source domain level effects, guard functions, target domain level effects.
Step 16:	Create a class "TestCase" and write it to the file *T*. Create three classes "PreCondition", "TestInput", and "ExpectedResult". Add these classes as subclass towards "TestCase" and write the related statements to file *T*.
Step 17:	Create two object properties "hasTestInput" and "hasPrecondition" and write these two object properties to file *T*.
Step 18:	Read the file *R* containing set of inference rules.
Step 19:	Apply the set of inference rules on file *T* using the generic rule reasoner, generate the test cases as per inference rules and print those test cases to the console.

Fig. 7 Partial view of the proposed rule-based reasoned implemented in Apache Jena

```
public static void main(String[] args) throws IOException
{
Scanner sint= new Scanner(System.in);
OntDocumentManager mgr=new OntDocumentManager();
OntModelSpec s=new OntModelSpec(OntModelSpec.OWL_MEM);
s.setDocumentManager(mgr);
OntModel m1=ModelFactory.createOntologyModel(s,null);
OntModel TestModel=ModelFactory.createOntogyModel(s,null);
TestModel.createOntology(NS1);
-------------------------------------------------- --------
-------------------------------------------------------
goalId=(objectid.asLiteral().getInt());
if(goalId==REQID)
{System.out.println("The objective of the requirement-ID"+
" "+goalId +" "+"is " +subjectg);
StmtIterator gdoeff=m1.listStatements(subjectg, extended,
(Resource) null);
while (gdoeff.hasNext()){
----------------------------------------------------------
StmtIterator csdoeffdr=m1.listState-
ments(cause_list_tr.get(j),DR,(Resource)null );
while(csdoeffdr.hasNext())
{Statement casdeffdr = csdoeffdr.next();
Resource objecteffdr=(Resource) casdeffdr.getObject();
cause_effect_list_dr.put(cause_list_tr.get(j),ob-
jecteffdr);}
effectsdr=cause_effect_list_dr.get(cause_list_tr.get(j));
effect_list_dr= new ArrayList<Resource>(effectsdr);
for(m=0;m<effect_list_dr.size();m++){
-----------------------------------------------------------
File f1 = new File(input0);
if (f1.exists()) {
List<Rule> rules = Rule.rulesFromURL("file:" + input0);
GenericRuleReasoner r = new GenericRuleReasoner(rules);
r.setOWLTranslation(true);r.setTransitiveClosureCach-
ing(true);
---------------------
StmtIterator effhapreconeff=testModel.listStatements(ef-
fect_list.get(i),hasPreCondition,(Resource) null);
while (effhapreconeff.hasNext()){------------------
```

Table 4 summarization of the users' goals and corresponding domain effects present in the case study described in Sect. 5.1

Users' goals	Corresponding domain effects
Framer's_Registration_Process	Soil_Sample_is_Rejected
	Soil_Sample_is_accepted
	Farmer's_Registration
	Message_sent_to_farmer
	Fee_is_collected
	Sample_Soil_Collected
	Fee_is_not_collected
Soil_Health_Card_Generation	Acknowledgement_to_the_farmer
	Soil_health_Card_is_Generated
Testing_of_Soil_Sample	Sample_Test_is_Accepted
	Test_Result_is_displayed
	Sample_Test_is_Rejected

Proposed Inference Rules for Test Case Automation

The proposed inference rules are specified based on causes, effects, transformation relationships, dependency relationships and different guard functions of ODRA. These rules are domain independent. Hence, they are applicable to different domains and applications. Thus, customized domain and application-based test case generation is possible through the proposed approach. These proposed inference rules are intended for mapping from the ODRA-based scenario towards test case ontology as specified in Table 1. Figure 5 has illustrated an example of the proposed rules. This example represents there are two different instances (*a*,*b*) of domain level cause. Both *a* and *b* are transferred towards a domain level effects *c* and related with each other using And guard function. Further, *a* and *b* both depends on

Table 5 Summarization of domain level causes and corresponding domain level effects and *DR*/*TR* relationships present in the case study described in Sect. 5.1

Domain specific causes	Corresponding domain effects and DR/TR relationship
Sample_Soil	Sample_Soil_Collected (*TR*)
Farmer	Sample_Soil_Collected (*TR*)
Physical_Verification_of_Sample_Soil	Soil_Sample_is_accepted (*TR*)
	Message_sent_to_farmer (*TR*)
	Sample_Soil_Collected (*DR*)
Registration_Officer	Soil_Sample_is_accepted (*TR*)
	Message_sent_to_farmer (*TR*)
	Farmer's_Registration (*TR*)
Check_fee_is_applicable_or_not	Soil_Sample_is_accepted (*DR*)
	Fee_is_collected (*TR*)
Inward_Number_generation	Fee_is_collected (*DR*)
	Farmer's_Registration (*TR*)
Lab_Code_no._assigned	Farmer's_Registration (*DR*)
	Sample_Test_is_Accepted (*TR*)
Analyst	Sample_Test_is_Accepted (*TR*)
	Test_Result_is_displayed (*TR*)
Soil_Testing_Officer	Test_Result_is_displayed (*TR*)
Soil_test_results_are_confirmed	Acknowledgement_to_the_farmer (*TR*)
	Soil_health_Card_is_Generated (*TR*)
	Test_Result_is_displayed (*DR*)

domain level effect *d* and *l*, respectively. Besides this, *c* is related with another domain level effect *i* through Require guard function. If all these conditions are met, then *a* and *b* become "TestInput", *d*, *l*, and *i* become "PreCondition" and *c* become "TestOutput" in the automated test case. Further, *c* will be related with *a* and *b* using "hasTestInput" object property. In addition, *c* will be related with *d*, *l* and *i* using "hasPrecondition" relationship.

To illustrate the rule specified in Fig. 5, let an example of a system that facilitates in online shipping of products. Upon getting the request of shipping of a product, at first, the system checks the stock for the availability of the product. If the product is in stock, the order is confirmed otherwise it is rejected. Next, if the order is confirmed, then the system will create an invoice, update the stock and ships the good to the customer after proper packaging. Table 2 has listed the causes and effects of this example. The CED graph for this example is illustrated in Fig. 6. Further, Table 3 has listed the causes and effects in this example; those are mapped with the instances in the rule specified in Fig. 5. Based on this mapping, the "TestInput", "TestOutput" and "PreCondition" class of proposed test case ontology will be populated for the test case, which is generated according to the rule. Table 3 also specifies the auto generated "TestInput", "TestOutput" and "PreCondition" for this specific test case.

Implementation of the Proposed Methodology

In this section, the proposed reasoner is implemented using Apache Jena. Apache Jena is a free and open-source Java framework for building semantic web and Linked Data applications. The framework is composed of different APIs interacting together to process RDF data. It supports processing of ontology expressed in OWL by giving access to a range of inference capabilities. Jena has several built-in *reasoners. Generic* rule reasoner is one, which can reason over an ontology specification based on users' defined rules. Those rules should be defined in Apache Jena rule syntax. The proposed inference rules are represented using Apache Jena rule syntax. Figure 7 has illustrated the partial view of the proposed reasoner implemented in Apache Jena.

Illustration of the Proposed Methodology Using Case Studies

In this section, the proposed methodology is illustrated using two case studies. The first one is related with soil testing management system. The second one is related with healthcare professional in rural area. Two case studies are used in order to demonstrate that the proposed approach can be applied on different domains. This is one important

Fig. 8 Partial view of the automated test cases generated through the proposed rule-based reasoner for the case study specified in Sect. 5.1

```
What is the requirement ID?
1
The objective of the requirement-ID 1 is http://www.seman-
ticweb.org/user/ontologies/2017/5/untitled-ontology-
134#Framer's_Registration_Process
'Test Case 1' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-
134#Fee_is_collected>  'Test Input:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Regis-
tration_Officer>  <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Inward_Number_generation>
'Expected Result' <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Farmer's_Registration>
'end'
'Test Case 2' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-
134#Soil_Sample_is_accepted>  'Test Input:' ---------------
--------- <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Fee_is_collected> 'end'
'Test Case 3' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Sam-
ple_Soil_Collected>  'Test Input:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Regis-
tration_Officer>  <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Physical_Verifica-
tion_of_Sample_Soil>  'Expected Result' <http://www.seman-
ticweb.org/user/ontologies/2017/5/untitled-ontology-
134#Soil_Sample_is_accepted> 'end'
'Test Case 4' 'Test Input:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-
134#Farmer>  <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Sample_Soil>  'Expected
Result' <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Sample_Soil_Collected>
'end'
'Test Case 5' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Sam-
ple_Soil_Collected>  'Test Input:' ------<http://www.seman-
ticweb.org/user/ontologies/2017/5/untitled-ontology-
134#Physical_Verification_of_Sample_Soil>  'Expected Re-
sult' <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Message_sent_to_farmer>
'end'
.../Jena Done/....
```

contribution of the proposed approach specified in the paper since most of existing approaches are domain specific.

Description and Implementation of the First Case Study

Let, a case study on soil testing management system. In this case study, a farmer brings soil sample in lab for testing. Upon checking the condition of soil, the sample is accepted or rejected by registration officer. If the soil sample is accepted, then registration officer checks if fees are applicable to the corresponding farmer or not. If fee is applicable, then fee is collected through either cash or back receipt. Next, sample is sent to lab for both fee waiver and payee farmer. A message is sent to the farmer's mobile for acceptance of the soil. An inward number is generated for the soil's sample. After that, farmer's registration is done. Next, lab code number is assigned to the sample along with the serial number of the soil being sent to the lab by soil testing officer. Then, sample comes to the analyst in lab. Soil is tested in the lab by analyst. If sample is valid, then readings are noted and entered in the system by analyst. Test result is displayed on the screen by soil testing officer. Soil Health Card is generated and stored. A message is sent to farmer's mobile regarding generation of Soil Health Card.

Table 6 summarization of the the users' goals and corresponding domain effects present in the case study described in Sect. 5.2

Users' goals	Corresponding domain effects
Visit towards the patient	Make all calls
	Gather all prepared materials
	Prepare injection
	No need of preparing injection
	Drive to patients' home
	Use of GPS
	No need of use of GPS
	Register the visit
Documents all patients' visit	Discuss with patients
	Prescribe medication
	No Need to prescribe medication
	Prepare documents of all patients' visit

From this case study, three users' goals are identified and those are mapped towards Goal concept of ODRA. Those three goals are "Farmer's Registration Process", "Testing of Soil Sample" and "Soil Health Card Generation". These goals are extended to several domain level effects. Table 4 has summarized the users' goals and corresponding domain effects. Further, several domain level causes are transferred or dependent towards/on these domain level effects. Table 5 has summarized this listing. Figure 8 has illustrated the auto-generated test cases for the goal "Framer's Registration Process". The proposed reasoner has generated the test cases along with required pre conditions, test input and expected result. However, the pre-condition segment is optional. Where the pre-condition is available, it is generated. For example, in Fig. 7, in the test case 1 and test case 4, the pre-condition is not generated. Further, different combinations of test data are identified in the auto-generated test cases and specified as "Test Input". Besides this, if domain knowledge is modified, then the ODRA specification is also modified accordingly, and thus the automated test cases are updated. In addition, the proposed methodology has supported test case representations for customized users requirements of different domain and applications based on users' goal id. This contribution of the proposed work facilitates in improving both users' requirements coverage and domain coverage. Thus, the proposed work in this paper has addressed the challenges mentioned in introduction.

Description and Implementation of the Second Case Study

Let, a case study on healthcare in rural area. In this case study, daily activities of a health professional in a rural area is described. Daily activities of a health professional is started with printing a to-do list for all patients visits during the day. The next activity is to make calls to different persons to co-ordinate work and activities. Health care professionals gather all the materials related to vital signs and/or measurement of patients. Further, they also check if there is any need of preparing injection or not. After that, the patient

Table 7 Summarization of domain level causes and corresponding domain level effects and *DR*/*TR* relationships present in the case study described in Sect. 5.2

Domain specific causes	Corresponding domain effects and DR/TR relationship
Print to do list	Make all calls (*TR*)
Health professional	Make all calls (*TR*)
Check for injection	Prepare injection (*TR*)
	Make all calls (*DR*)
Print vital signs/measurements	Gather all prepared materials (*TR*)
	Prepare injection (*DR*)
Call patient to notify	Gather all prepared materials (*DR*)
	Drive to patients' home (*TR*)
Check for new patient	Use of GPS (*TR*)
	Drive to patients' home (*DR*)
Visit to the patients home	Use of GPS (*DR*)
	Register the visit (*TR*)
Health professional	Register the visit (*DR*)
	Discuss (*TR*)
Patients	Register the visit (*DR*)
	Discuss (*TR*)
Check for need of prescribing medication	Prescribe medication (*TR*)
	Discuss (*DR*)
Write notes of all things	Prescribe medication (*DR*)
	Prepare documents of all patients' visit (*TR*)

Fig. 9 Partial view of the automated test cases generated through the proposed rule-based reasoner for the case study specified in Sect. 5.2

```
What is the requirement ID?
2
The objective of the requirement-ID 2 is http://www.seman-
ticweb.org/user/ontologies/2017/5/untitled-ontology-
134#Documents_All_Patients'_Visit
'Test Case 1' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Pre-
scribe_Medication>  'Test Input:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Tak-
ing_Notes_of_All_Things>  'Expected Result' <http://www.se-
manticweb.org/user/ontologies/2017/5/untitled-ontology-
134#Prepare_Documents_of_all_patients'_visit> 'end'
'Test Case 2' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Dis-
cuss_with_Patients>  'Test Input:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-
134#Check_for_Need_of_Prescribing_Medications>  'Expected
Result' <http://www.semanticweb.org/user/ontolo-
gies/2017/5/untitled-ontology-134#Prescribe_Medication>
'end'
'Test Case 3' 'Pre-Condition:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Regis-
ter_the_Visit>  'Test Input:' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-
134#Health_Professional>  <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Pa-
tients>  'Expected Result' <http://www.semantic-
web.org/user/ontologies/2017/5/untitled-ontology-134#Dis-
cuss_with_Patients> 'end'
.../Jena Done/....
```

is informed that the healthcare professionals are coming. If the patient is new, then a GPS can be used to navigate. The healthcare professional registers their visit. If there is need, then medications are prescribed towards the patients. Healthcare Professionals write note of all the related things. All patient visits must be documented in the medical record system.

This case study has two goals those are mapped towards Goal concept of ODRA. The first is “Visit towards the patient”. The second is “Documents all patients’ Visit”. Table 6 has summarized the users’ goals and corresponding effects. Table 7 has summarized the listing of domain level causes, corresponding domain level effects and DR/TR relationships. Figure 9 has illustrated the auto-generated test cases for the goal “Visit towards the patient”.

Conclusion and Future Work

In existing literatures, automated test case generation for black box testing is not considered attentively. This paper has addressed this issue and devised a rule-based reasoner for auto generation of test cases from an ontology-based requirements specification. The proposed reasoner has generated test cases specifically for black-box testing. It is devised in Apache Jena. The contributions of the proposed work are to facilitate in (1) specification of domain independent inference rules those aid in test case generation for different domains and applications, (2) auto upgrade of test cases as per modification of domain knowledge, (3) auto identification of pre-conditions related to a test case if present, (4) auto identification of different combinations of similar test data and (5) improvement of users’ requirements coverage and domain coverage.

Future work will include, automated test script generation for auto-generated test cases from the ontology driven requirement facets of Applications in specific. Further, evaluation of proposed testing strategy and test cases will be also a significant future work. In addition, adoption of proposed methodology in modern technology, such as Internet of Things, cloud-based applications will be a prime focus.

Conflict of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Dadkhah M, Araban S, Paydar S. A systematic literature review on semantic web enabled software testing. J Syst Softw. 2020. https://doi.org/10.1016/j.jss.2019.110485.
2. Wnuk K, Garrepalli T. Knowledge management in software testing: a systematic snowball literature review. Informatica Softw Eng J. 2018;12(1):51–78.
3. Tarasov V., Tan H., Adlemo A. (2019) Automation of software testing process using ontologies. In: Proceedings of the 11th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD), vol. 2, pp. 57–66.
4. Tarasov V., Tan H., Ismail M., Adlemo A., Johansson M. (2016) Application of Inference Rules to a Software Requirements Ontology to Generate Software Test Cases. In: OWL: Experiences and Directions – Reasoner Evaluation: 13th International Workshop, OWLED 2016, and 5th International Workshop, ORE 2016, Bologna, Italy, November 20, 2016, Cham: Springer, 2017, pp. 82–94.
5. Olajubu O., Ajit S., Johnson M., Turner S., Thomson S., Edwards M.: Automated test case generation from domain specific models of high-level requirements. In: RACS: Proceedings of the 2015 Conference on research in adaptive and convergent systems, October pp.505–508 (October 2015).
6. de Souza E'F, Falbo RDA, Vijaykumar NL. Knowledge management initiatives in software testing: a mapping study. Inform Softw Technol. 2014;57(2014):378–91.
7. Guarino N, Oberle D, Staab S. What is an ontology? In: Staab S, Studer R, editors. Handbook on ontologies. 2nd ed. Berlin: Springer-Verlag; 2009. p. 1–17.
8. Banerjee S, Sarkar A. Domain-specific requirements analysis framework: ontology-driven approach. Int J Comput App. 2019;2019:1–25.
9. Horridge M. A.: Practical guide to building OWL ontologies using Protégé 4 and COODETools. Edition 1.3. The University of Manchester [Internet]. 2011, March 24, https://mariaiulianadascalu.files.wordpress.com/2014/02/owl-cs-manchester-ac-uk_-eowltutorialp4_v1_3.pdf, last accessed on 2020/09/06.
10. Apache Jena, https://jena.apache.org., Accessed on 2020/09/04.
11. Kanstrén T. (2013) A review of domain-specific modelling and software testing. In: Proceedings of Event8th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2013), pp. 51–56
12. Haq S. U., Qamar U. (2019) Ontology Based Test Case Generation for Black Box Testing. In: Proceedings of the 2019 8th International Conference on Educational and Information Technology (ICEIT 2019), pp. 236–241
13. Banerjee S, Sarkar A. A requirements analysis framework for development of service oriented systems. SoftwEngNotes ACM-Sigsoft. 2017;42(3):1–12.
14. Nasser V. H., Du W., MacIsaac D. (2010) An Ontology-based Software Test Generation Framework. In: Proceedings of Software Engineering and Knowledge Engineering (SEKE), pp. 192–197
15. Sneed H. M., Verhoef C. (2013) Natural language requirement specification for web service testing. In: 15th IEEE International Symposium on Web Systems Evolution (WSE), Eindhoven, pp. 5–14
16. Wang Y., Bai X., Li J., Huang R. (2007) Ontology-Based Test Case Generation for Testing Web Services. In: 8th International Symposium on Autonomous Decentralized Systems (ISADS'07), Sedona, AZ, pp. 43–50
17. Nguyen C. D., Perini A., Tonella P. (2008) Ontologybased test generation for multiagent systems. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems (AAMAS), 3:1315–1320
18. de Souza EF, de Falbo RA, Vijaykumar NL. ROoST: Reference Ontology on SoftwareTesting. Appl Ontol. 2017;12(1):59–90.